IChemE

Trans IChemE, Vol 77, Part B, November 1999

HAZID, A COMPUTER AID FOR HAZARD IDENTIFICATION
2. Unit Model System

S. A. McCOY (ASSOCIATE), S. J. WAKEMAN, F. D. LARKIN (ASSOCIATE), P. W. H. CHUNG, A. G. RUSHTON (MEMBER) and F. P. LEES (FELLOW)†

Department of Chemical Engineering, Loughborough University, Loughborough, UK

The hazard and operability, or HAZOP, study is a prime method for the identification of hazards on process plants. This is the second in a series of papers which describes progress in the emulation of hazard identification in the style of HAZOP. The work reported is embodied in a computer aid for hazard identification, or HAZOP emulator, HAZID. The HAZID code is one of a suite of codes developed as part of the STOPHAZ project. The present paper describes the unit model system.

Companion papers describe: an overview of HAZID, with an account of HAZOP and HAZOP emulation, and of the issues underlying it; the fluid model system and evaluation of consequences; the evaluation and improvement of HAZID using case studies and other methods; some development topics. Conclusions from the work are given in the final paper.

Keywords: hazard identification; HAZOP; computer-assisted hazard identification; qualitative modelling; signed direct graphs.

INTRODUCTION

This paper is the second in a series on computer-aiding of hazard identification and describes the system for modelling of the plant equipment, or units. The content of the other papers is outlined in the first paper[1].

The unit models are used in the HAZOP emulation system, AutoHAZID. In this system the plant is described as a network of interconnected units and each unit is described in terms of a model based on signed directed graphs (SDGs).

This paper firstly describes the hierarchical, inheritance-based, framework for organizing the unit models in AutoHAZID. A description of the structure of process unit models is then given, concentrating on the different slots used. The arcs used to model fault propagation, which form the heart of the model in AutoHAZID, are then described. Brief mention is given to failure frequency for faults, conditional faults and consequences, and consequence evaluation. The first is a feature provided but not used in AutoHAZID and the other features are described fully in the next paper, rather than this one. A description of the Model Generation Tool (MGT) is given, followed by discussion of the unit model templates upon which the MGT relies. The paper concludes by describing how instrumentation and control units are modelled.

UNIT MODELS AND MODEL LIBRARY

The unit models are organized into a hierarchical structure and placed in a model library. Each model within the library is a distinct entity, but the hierarchical structure of the library supports the inheritance of characteristics between models. The names of the models present in the unit model library and its hierarchical structure are shown in Figure 1.

Inheritance

Each unit model (child) is defined as being of a particular type (parent). The child model inherits, in addition to its own information, all the information contained within the parent model. Consider the unit model pump in Figure 1: the units centrifugalPump and pdPump are child units of the parent unit pump. The parent model contains information which is applicable to all its children. For example, loss of power to a pump will result in reduction in outlet pressure. On the other hand, a child model contains information of relevance to itself but not to other children of its parent. For example, loss of power is a potential cause of reverse flow through a centrifugal pump (centrifugalPump) but not through a positive displacement pump (pdPump).

The advantage of using inheritance is that it allows the unit models to be kept small and it makes the creation of new models easier, as only that information which distinguishes a child from its parent must be defined.

Unit Model Structure

The basic structure of a process unit model is as shown below. Models for instrumentation and control are somewhat different and are treated separately. The model consists of a declaration frame which contains a number

† Frank Lees (1931–1999), Professor of Plant Engineering, Loughborough University.

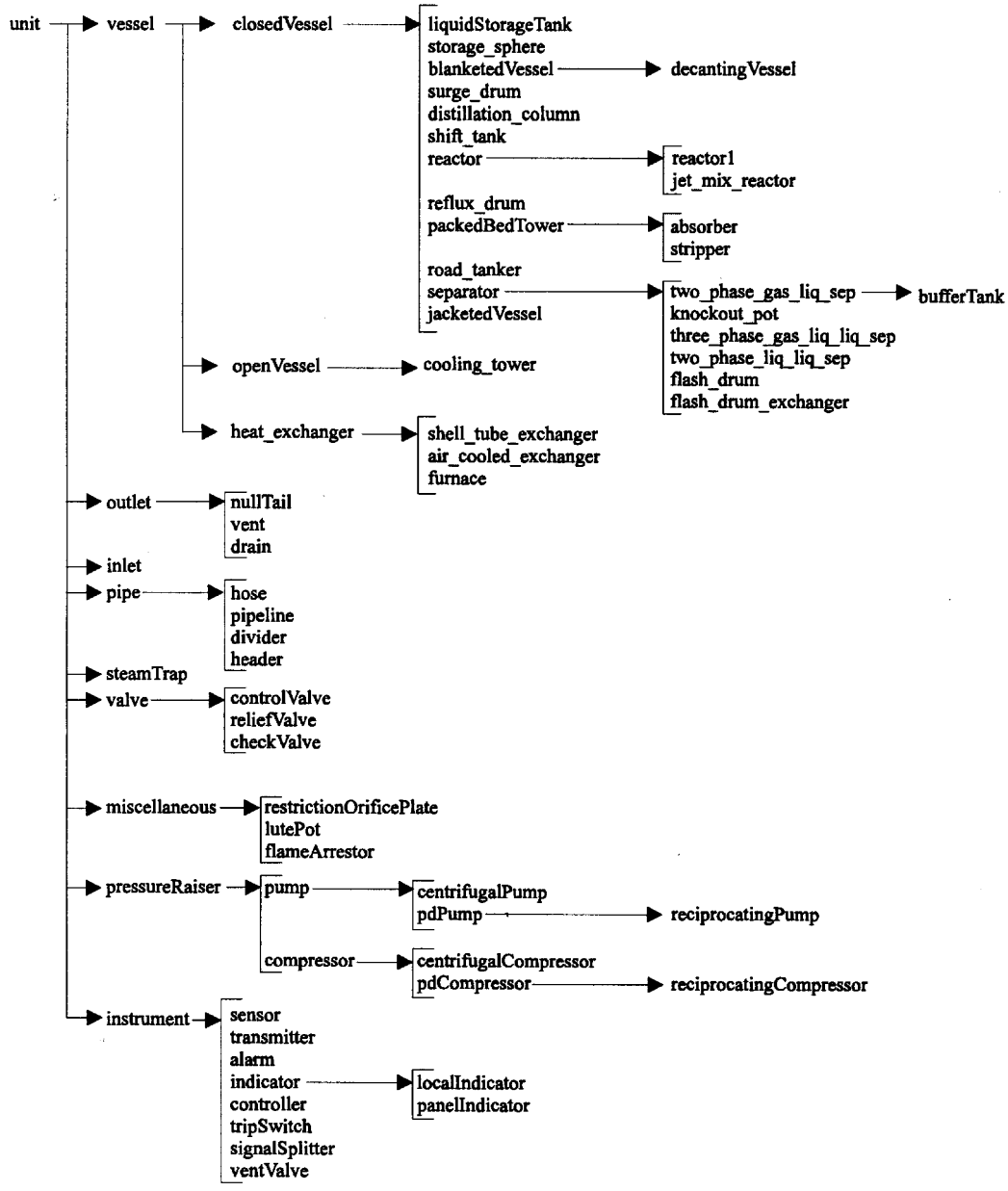

Figure 1. Hierarchy of the unit model library.

of optional lines (slots), each of which defines a particular type of information.

```
frame(unit_name isa parent_name,
        inports,
        outports,
        unitports,
        must_connect,
        comp_connections,
        propLinks,
        attributes,
        script,
        conditionLinks,
        other_slots).
```

Inlet, outlet and unit ports

The inports, outports and unitports slots are used to specify the names of the process inlet, outlet and internal ports of a unit respectively. If a port name is not specified in any of these slots, or inherited from another model, it will not be recognized as a port of the relevant type if referred to later in the model.

Internal ports are used to refer to distinct regions within a unit. Examples of such regions are the liquid and vapour phases in a tank or the shell side and the tube side of a shell and tube heat exchanger. Each of the regions in these examples must have a unit port name assigned to it in order that it can be referred to in the propagation reasoning. Consider a deviation in temperature of a vapour flow into the vapour space of a liquid storage tank: the high inlet temperature will lead to an increase in the temperature of the vapour space but not to any significant increase in the temperature of the liquid. The ability to make this distinction between the different temperatures within a unit depends on the definition of different regions, each with its own name.

Obligatory connections

There are certain ports on a process unit each of which must be connected to another unit for it to operate, e.g. the inlet and outlet ports of a pump. The same assertion does not apply for some other ports, e.g., a drain port on a pump.

The must_connect slot contains a list of inlet and outlet port names. If there is any port in this list which is not connected to another unit when the model is used in a particular plant description, then on loading a warning will be produced. This provides a check on the correctness of the plant description.

The comp_connections slot is used to define the flow connections within a unit. It serves to determine the flow paths within a process. Consider a shell and tube heat exchanger: the tube side inlet port (tubeIn) is connected to the tube side outlet port (tubeOut) and likewise for the shell side:

```
comp_connections info [[tubeIn,
tubeOut],[shellIn,shellOut]]
```

No connection is defined between the shell and tube sides of the exchanger and there should be none.

Propagation links

The propLinks slot is used to define the qualitative relationships between faults, deviations and consequences within an item of equipment. This is the most important part of a unit model and further details are given below, in the discussion on fault propagation.

Attributes

Unlike the other slot types in the unit models, attributes may not have a list of items as their value; they are limited to having one symbol as their value (e.g., '`status is on`' for a pump). They are used to define simple properties of process units for sub-typing or status information. If a value is given to an attribute in a unit model, this is treated as the default value for that attribute; the default can be over-written when an instance of that model is used in a plant model.

In conjunction with the conditionLinks slot, the typing information provided by attributes allows a model to be made more specific, dependent on the values of its various attributes. For example, a pipeline may have an attribute for construction, with possible values of welded, flanged, etc., and another attribute for lagging, with values of lagged or unlagged. Each attribute contributes to specializing the unit model in a certain way, when it is used in a plant model.

Scripts

Script slots are used to add one or more templates to a unit model. Adding a template to a model in effect adds a parcel of propagation relationships which represents a particular physical entity or phenomenon in a unit. Using scripts is therefore an indirect method of adding propagation arcs. Templates are considered further below.

Condition links

As mentioned above, conditionLinks slots allow an otherwise quite generic unit model to be specialized, depending on the values of its attributes when instances of that model are used in a plant model. The conditionLinks slot associates an attribute-value pairing with a number of slots to be added to the unit model in the event that the given attribute has the given value. The values of the attributes are only matched when an equipment model is instantiated in a plant, so that the conditionLinks slot in a unit model represents a number of possible model variations, or 'sub-types'.

Other slots

In addition to the slots listed above, various other slots are used to provide miscellaneous information about the temperatures, pressures and compositions of the fluids in the unit, design pressures and temperatures, operational state of the equipment, etc. These are used particularly in the fluid model system to provide the information needed to validate fault-consequence scenarios.

FAULT PROPAGATION

A unit model contains a number of different types of information about a process unit. By far the most important is the set of qualitative propagation relationships which lie at the heart of AutoHAZID's inference technique. A unit model gives a qualitative definition of:

1. The propagation of deviations through a unit.
2. The faults which occur within a unit and their effect on the process variables (fault initiation).
3. The consequences of deviations and faults within a unit (fault termination).

Each of these relationships is represented by an arc in the SDG part of the unit model. The arcs are typically declared using propLinks slots or in template definitions. There are four types of arc, representing the following types of propagation link:

- Deviation $\rightarrow$ Deviation
- Fault $\rightarrow$ Deviation
- Deviation $\rightarrow$ Consequence
- Fault $\rightarrow$ Consequence

Deviation-Deviation Propagation

A deviation-deviation arc defines the effect of a deviation of one process variable in the unit on another. The syntax of such an arc is:

```
arc([port1,variable1],influence,
[port2,variable2])
```

The arc can be interpreted as: a deviation in variable1 at port port1 will have an effect on variable2 at port port2, the sign of this effect being represented by the sign of the integer influence (usually +1 or —1). Specifically:

```
arc([port1,variable1],-1,[port2,
variable2])
```

means that an increase in variable1 at port port1 results in a decrease of variable2 at port port2 and a decrease in variable1 at port port1 results in an increase of variable2 at port port2.

The direction of influence is always that a deviation of the left-hand variable (variable1 in the example just given) affects the right-hand variable (variable2 in the example). A deviation in the right-hand variable has no effect on the

Table 1. Values of the integer influence in fault propagation arcs.

`arc([port1,X],N,[port2,Y])`	
Influence Value (N)	Interpretation
+1	high X → high Y low X → low Y
−1	high X → low Y low X → high Y
+2	high X → high Y
−2	high X → low Y
+3	low X → low Y
−3	low X → high Y

left-hand one, unless this effect is explicitly stated in another arc.

The possible values that the integer influence can take are shown and explained in Table 1.

Fault-Deviation Propagation

A fault-deviation arc defines the effect of a possible fault on a process variable in the unit. The syntax of such an arc is:

```
arc([fault,'fault descriptor'],
influence,[port,variable])
```

The first argument [fault, 'fault descriptor'] has two elements: the first specifies that the entity concerned is a fault, the second is the fault descriptor string which is displayed when the fault is described in the output report.

The argument 'influence' is an integer which specifies whether the fault causes an increase in the variable (influence is 1) or a decrease (influence is −1). This argument can have no other values in fault-deviation propagation arcs.

Deviation-Consequence Propagation

A deviation-consequence arc defines the possible consequence of a process variable deviation within the unit. The syntax of such an arc is:

```
arc([deviation,[dev_label,port]],1,
[consequence,'consequence descriptor'])
```

The first part of the argument [deviation, [deviation label, port]] specifies to the system that the entity concerned is a deviation, the second part specifies the type of deviation, e.g., moreTemp or lessFlow, and the port of the unit at which the deviation occurs.

The consequence block [consequence, 'consequence descriptor'] has two parts: the first specifies the entity concerned as a consequence and the second is the consequence descriptor string which is displayed when the consequence is described in the output report.

Fault-Consequence Link

It is sometimes desirable to link a fault to a consequence directly rather than via a deviation. For example, the consequence of a leakage fault such as flammable release is related to the fault itself rather than to any deviation caused by it. A fault-consequence arc makes it possible to specify such a direct link. The syntax of such an arc is:

```
arc([fault,'fault descriptor'],1,
[consequence,'consequence descriptor'])
```

The fault and consequence blocks in the above have the same format as described for the other arc types.

FAILURE FREQUENCY

The unit model scheme provides the facility to assign to each fault a frequency of occurrence. This is done at the level of the unit model library rather than of the unit model itself. The value should be an integer between 1 and 5, 1 being the lowest frequency and 5 the highest. These frequency values can then be used to rank faults by frequency. An example of the definition of a failure frequency is:

```
arc([fault,'fails closed'(5)],1,[out,
noFlow])
```

This fault relates to the failure of a control valve to the closed position, resulting in no flow at the outlet port (out) of the valve. The fault has been assigned the highest frequency of 5, indicating that it is a very common type of failure.

Although this facility has been provided, no use has been made of it in the current implementation, in which consequence evaluation is based instead on a set of rules. It is envisaged, however, that a more refined evaluation scheme is likely to utilize frequency estimates. It is also recognized that the definition and creation of a suitable frequency database is not a trivial undertaking.

CONDITIONAL FAULTS, CONDITIONAL CONSEQUENCES, AND CONSEQUENCE EVALUATION

The unit models are intended to be as generic as possible. However, many of the faults and consequences in a unit model may not be feasible in a particular plant. The properties of the fluids are the main determining factor. The use of conditions, dependent on fluid properties, allows the feasibility of a fault or consequence to be determined. If on resolution of the condition the fault or consequence is found not to be valid, it is excluded from the output report. This is the method used by the fluid model system for validating the results of HAZOP emulation.

Even after this condition validation procedure has been applied, there generally remain a large number of consequences only some of which are of real interest. It is necessary therefore to perform an evaluation of these consequences in order to rank them in order of severity.

The topic of conditional faults and consequences, as well as that of consequence evaluation, are covered in more detail in paper three of this series[2].

MODEL GENERATION TOOL

If there is a requirement for a unit model which is not in the unit model library, it is necessary to create one. For this, use is made of the Model Generation Tool (MGT). The MGT can also be used to create a new model by modifying one which is already in the library. The MGT is designed for

the generation of a model for any unit which has a chamber, which covers most cases of interest. It is not intended for the generation of very simple pipe-type models.

The MGT provides an interactive interface which takes the user through a question and answer session in which a sequence of questions are asked which elicit the information needed to build up the elements of the unit model. The MGT builds a model from a number of building blocks: chambers, phases and ports. The specification of the building blocks focuses on mechanical and phase interfaces.

Mechanical interface

The term mechanical interface refers to a structural barrier separating two sections, or chambers, within a unit. The barrier could be a baffle separating two bodies of liquid or a heat transfer interface such as heat transfer tubes/coils or a vessel jacket. Baffles and heat transfer interfaces are the only two mechanical interfaces supported by the MGT. Containment (from atmosphere) is implicit and is not specified explicitly in the model generation process.

Phase interfaces

The term phase interface refers to an interface between two fluid phases. The two fluid interfaces supported are vapour-liquid and liquid-liquid.

Chamber

The term chamber refers to a distinct region within a unit. Chambers are structural entities and are not concerned with contents, i.e., the phase(s) of the fluids concerned. A unit must have at least one chamber.

Ports

Ports specified by the user are the inlet and outlet connections of a unit. In the MGT a port is considered to be connected to a particular phase within a particular chamber.

Internal ports are also utilized. These are generated automatically based on the data entered by the user. An internal port is generated for each phase of a chamber and assigned the phase's name (as specified by the user). The exception to this is when only one phase exists in a chamber, in which case the name assigned to the internal port is that of the chamber.

Phase

The term phase refers to a distinct fluid phase within the unit. The phases supported by the MGT are gas/vapour and liquid.

UNIT MODEL TEMPLATES

Many items of equipment on process plant come in a number of variants and this is necessarily reflected in the unit models. In such a case a new variant of the unit model may differ only marginally from an existing one. In particular, tanks and vessels exhibit a large number of variations, many relatively minor. Furthermore, there are many structural features and physical phenomena which occur commonly in plant equipment.

The configuration of new models, therefore, tends to involve a degree of repetition so that there is scope for the definition of features which can be described by a 'template', embodied in a set of arcs which can be imported whenever this feature is required in a model. As indicated above, templates can be used not only for complete units such as a vessel but also for structures and phenomena. Application of the template concept to the latter constitutes a powerful extension of the unit modelling method.

Consider the similarities between a liquid storage tank and a liquid-vapour separator:

- Both have a containing wall
- Both have a body of liquid

Each of the commonalities will have an associated set of phenomena. For example, a containing wall can leak or be subjected to an external fire, whilst a body of liquid can freeze or boil.

Once the common structures and phenomena were recognized, the arcs associated with them were developed and grouped together, as templates. One or more templates can be included in a unit model using a script slot which specifies a call for the inclusion of the pertinent bundle of arcs.

There are a number of benefits to be gained by the use of templates:

1. The unit model is much more concise.
2. The maintenance of existing unit models is simplified.
3. The creation of new models is simplified.

Using templates, the unit models become more concise because a large group of arcs can be represented in the model by a single line. The maintenance of existing models is simplified, because if modifications affecting many models are required, only the templates and not the individual models need to be changed. The creation of new models is simplified, because many of the features required can be obtained by importing the appropriate templates.

Templates in model building for AutoHAZID are the analogue of subroutines in conventional computer programs.

Template Structure

The whole point of templates is that they are applicable to a wide range of units. In practice this means that information must be passed to them as arguments, in much the same way that arguments are passed to functions in high level programming languages, such as C++.

A template is a collection of arcs and an arc usually requires a port name. A unit model has a set of port names which are specific to that unit. The use of a generic template means that it is necessary to pass these specific port names to the template as arguments in order to generate the specific arcs required.

The arcs present in a template are structurally identical to those which might appear in the unit models in the unit model library. The only difference is that instead of referring to entities specific to a particular unit they refer to argument names. As an illustration of the syntax used to define templates, consider a cut-down template for a particular port type, inPort1:

```
template(inPort1(X,Y,L),
 [
   arc([X,pressure],1,[L,pressure]),
   arc([L,pressure],1,[X,pressure]),
```

```
    arc([X,pressure],1,[X,flow]),
    arc([L,pressure],-1,[X,flow]),
    arc([Y,resistance],1,[X,pressure]),
    arc([Y,resistance],-1,[L,pressure]),
    arc([Y,resistance],-1,[X,flow])
  ]).
```

The reference to this template in a unit model would occur in a script slot, and look something like:

```
script info [inPort(in1,in1,liquid),...]
```

where in1 is the name of the port being modelled and liquid is the name of an internal port in the unit.

In this example the arcs in the template will be added (in the system's memory) to the unit model under construction with the following substitutions:

- X is replaced by *in1*
- Y is replaced by *in1*
- L is replaced by *liquid*

The first templates were developed for features of pipes and valves. Later templates modelled the structural features within vessels. A rough categorization of templates is therefore made, into vessel templates and in-line templates.

Vessel Templates

As the name implies, the vessel templates are of relevance to tank- and vessel-type units such as storage tanks, separators, columns, etc. Such units can be structurally decomposed into the following elements, for each of which there are numerous sub-types:

1. Ports.
2. Phases.
3. Phase interfaces.
4. Structural interfaces.

It can be imagined that starting with a vessel wall and adding the correct combination of each of the above elements almost any vessel type can be built up.

The vessel templates largely mirror the structures used in the question sequence in the Model Generation Tool. The templates and the MGT were developed in parallel—each answer in the MGT question sequence usually leads to the addition of one or more templates to the unit model.

The propagation of process variable deviations is treated in the port and interface templates.

Ports

The templates for ports model the relationship between all the relevant process variables in the stream attached to the port and those in the inside of the vessel There are also some generic faults and consequences within the port templates.

There is a wide range of port templates, which may initially seem odd since a port is structurally simply an aperture in a vessel wall. However, there are a number of factors relating to the propagation relationships for a port which must be considered:

1. Whether the port is an inlet or an outlet.
2. The normal/expected phase of the fluid passing through the port.
3. The connection point of the port (above or below the normal liquid level).
4. The phases normally in the vessel.
5. Whether or not the vessel is open to the atmosphere.

Phases

There are templates to represent bodies of liquid and of vapour in a tank, liquid_level and vapour_space, respectively. Here, liquid_level is an historic label, which with hindsight is not the most appropriate and could more consistently be replaced by liquid_space. The phase templates contain mainly the fault and consequence relationships and do not usually define variable propagation relationships.

Phase interfaces

The templates which model phase interfaces define the relationship between process variables on either side of the interface, e.g., an increase in the temperature of the liquid in a vessel causes an increase in the temperature of the vapour. The phase interface templates mostly deal with variable propagation and only in rare cases with faults and consequences. The phase interfaces for which templates have been created are liquid-vapour and liquid-liquid.

Structural interfaces

A structural interface is a physical barrier which separates two regions of a unit. Those currently modelled by templates are baffles and heat transfer interfaces.

Baffle templates, of which there are several, model the propagation of fluid flow across a baffle. These templates are needed because of the alternative arrangements for liquid overflow/underflow and vapour return flows which occur.

The heat transfer interface template (HtInterface) models an interface separating two regions of a unit, between which heat transfer is occurring. This template is not intended to be used on its own but in conjunction with other related templates, as described below.

In-line Templates

In-line templates model the relationships of the process variables between two points in a line, e.g., a pipe or a valve. The relationships are quite different from those for vessels. In addition, in-line templates also cover some phenomena such as leaks.

Miscellaneous Templates

There are a number of templates which do not fall neatly into either the vessel or in-line categories. Of these only those developed for modelling heat transfer between two regions of a unit need further explanation:

1. coldConduitHeatTransfer(X).
2. hotConduitHeatTransfer(X).
3. HtInterface(X, Y).
4. hotPoolHeatTransfer(X).
5. coldPoolHeatTransfer(X).

A combination of three of these templates is required to build up the relationships necessary for a particular heat transfer situation: one representing the interface (3); one representing the heat source (2 or 4); and one representing the heat sink (1 or 5).

In the above examples a distinction is made between heat

Table 2. Slots for instrument and control units.

Slot name	Slot content
outSignalPorts	List of names of ports through which a signal leaves a unit
inSignalPorts	List of names of ports through which a signal enters a unit
unitports	List of internal ports
sensedVariable	Reference to the process variable sensed by a unit (applicable to sensors and indicators only)
sigPropLinks	List of signal propagation arcs

transfer to/from conduits and pools. A conduit refers to the regions inside a tube or coil where flow is a consideration for heat transfer—the shell side of an exchanger is considered to be a conduit as flow affects heat transfer. Pool refers to a relatively static region of gas or liquid, e.g., a body of liquid in an evaporator vessel.

INSTRUMENTATION AND CONTROL UNITS

The treatment of instrumentation and control units is significantly different from that of process units. Like process units, instrumentation and control units have a number of allowable slots within their models. These slots are shown in Table 2.

Signal Propagation through Instrumentation and Control Units

The propagation of a signal between the inlet(s) and outlet(s) of instruments is defined using arcs in sigPropLinks slots. For example:

```
sigPropLinks info [sigArc(sigIn,
noCondition,sigOut]
```

where sigIn and sigOut are signal port names.

The sigPropLinks slot is similar to the propLinks slot for process units except that it defines propagation of signals rather than of deviations in process variables. The structure of the arcs for the propagation of signals is quite different from that used for process variable propagation.

The sigArc line defines how a deviation in a signal at a port represented by the first element affects the signal or variable represented by the last element. The second element in the structure is the condition which must be satisfied for the relationship to hold.

For units such as sensors, indicators and signal splitters where the signal is simply transmitted through the unit there is usually no condition (specified as noCondition). There are other units such as alarms and control valves where a specific deviation must occur before the unit activates. The deviation of relevance is specific to the plant concerned and therefore no condition is required in the unit model library, i.e., the arc is specified by default with the condition noCondition. The pertinent condition for a particular plant will exist in the plant description file which represents the plant concerned. For example, consider the following entry from a plant description file (not a unit model file):

```
instance(pcv1 isa controlValve,
  [ outports info [out is [valve19,in]],
    sigPropLinks info [
      sigArc(sigIn,[pcv1,out,pressure,
        -1],[self,aperture,1]),
      sigArc(sigIn,[pcv1,out,pressure,
        1],[self,aperture,-1])
    ]
  ]).
```

The condition in this case is a deviation in the outlet pressure from the control valve pcv1. The above example shows that the last element of a sigArc can be an action as well as a port. The last element is a port name if the unit merely propagates the signal.

The first sigArc in the example is interpreted as follows: a signal at the signal port sigIn (specified in the first element) will cause an increase (defined by the 1 in the third element) in the aperture of the valve if the deviation being propagated causes a reduction (defined by the –1 in the second element) in the pressure at the valve outlet.

REFERENCES

1. McCoy, S. A., Wakeman, S. J., Larkin, F. D., Jefferson, M. L., Chung, P. W. H., Rushton, A. G., Lees, F. P. and Heino, P. M., 1999, HAZID, a computer aid for hazard identification. 1: The STOPHAZ package and the HAZID code: an overview, the issues and the structure, *Trans IChemE, Part B, Proc Safe Env Prot*, 77 (B6):?
2. McCoy, S. A., Wakeman, S. J., Larkin, F. D., Chung, P. W. H., Rushton, A. G., Lees, F. P. and Heino, P. M., 1999, HAZID, a computer aid for hazard identification. 3: The fluid model and consequence evaluation systems, *Trans IChemE, Part B, Proc Safe Env Prot*, 77 (B6):?

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of the Commission of the European Community (Esprit Project 8228: STOPHAZ), their colleagues at Loughborough, and the contributions of many personnel from their partners in this project: ICI, Snamprogetti, SFK, Intrasoft, TXT, Aspentech, Bureau Veritas, Hyprotech and VTT. Paul Chung acknowledges the support of BG and the Royal Academy of Engineering through a Senior Research Fellowship.

ADDRESS

Correspondence concerning this paper should be addressed to Professor P. W. H. Chung, Department of Computer Science, Loughborough University, Loughborough LE11 3TU, UK.

Dr S. A. McCoy is now at the Department of Engineering Mathematics, University of Bristol, Queens Building, University Walk, Bristol BS8 1TR, UK.

The manuscript was received 22 April 1998 and accepted for publication after revision 27 October 1999.